Reflection in Java, CORBA und JacORB

Gerald Brose

Institut für Informatik

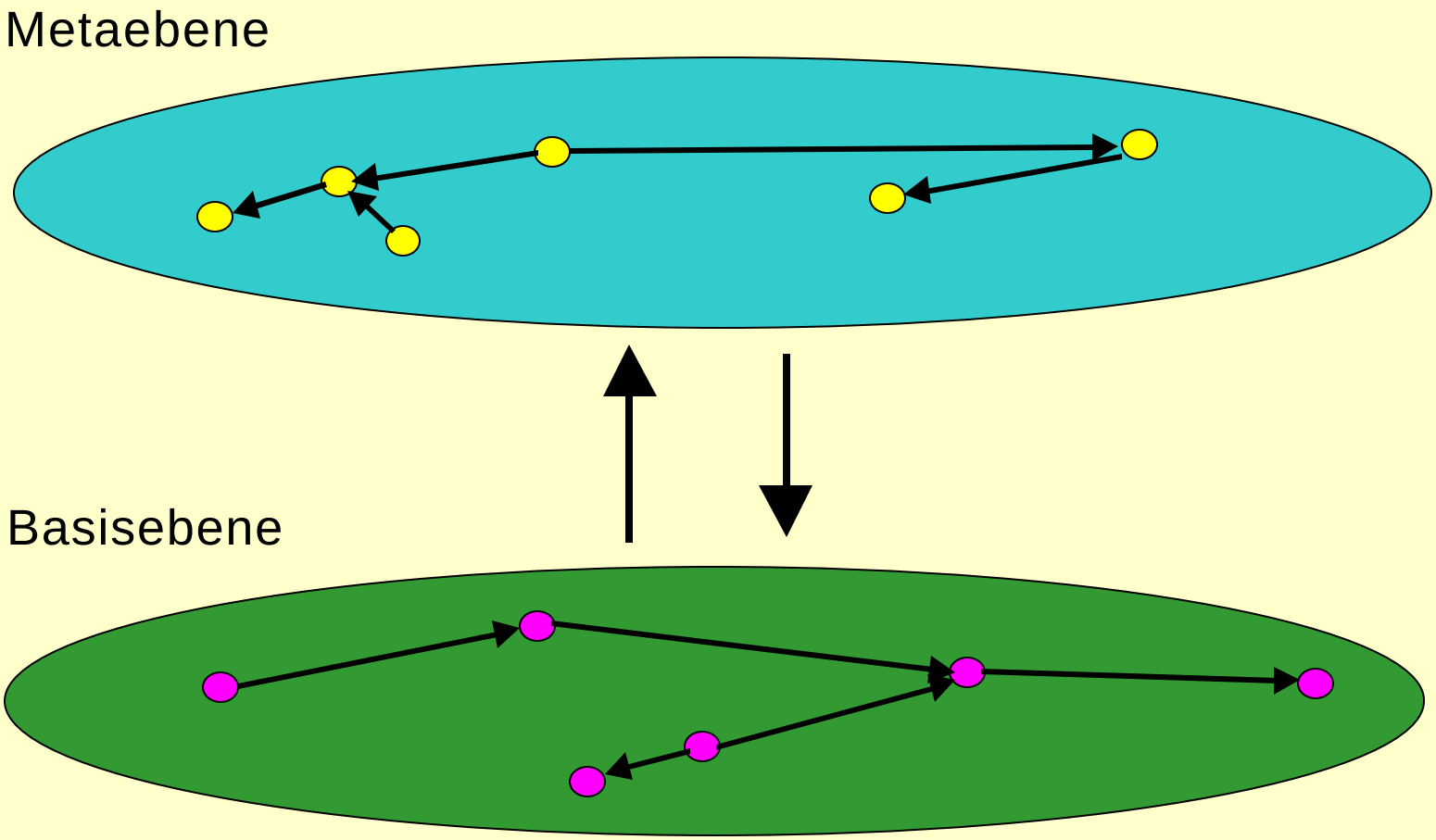
Freie Universität Berlin

Java-Informations-Tage 1998, Frankfurt/Main

Überblick

1. Reflection
2. Reflection in CORBA?!
3. Java
4. JacORB

1. Reflection



Reflection: Wozu?

Offenheit für *Änderungen*

- räumlich: Konfiguration
- zeitlich: Funktionalität
- technisch: Implementierung

Abstraktion

- Trennung funktionaler und nicht-funktionaler Aspekte (AOP)

Klassifikation von Reflection

- Wann geschieht Reflection?
- Was wird in der Metaebene “verdinglicht” (reifiziert)?
- Wie sieht die Schnittstelle zur Metaebene aus? (MOP/MLI)
- Wie wird in die Metaebene verzweigt?

Computational Reflection

- Meta-Objekte für *Berechnungs*einheiten
  - *structural* reflection: Inspektion
  - *behavioral* reflection: Modifikation
- “Was wird reifiziert?” (J. Ferber, 1989)
  - *meta-class model: Klasse*
  - *specific meta-object model: Objekte*
  - *meta-communication model: Aufrufe*

2. Reflection in CORBA?!

Computational Reflection:

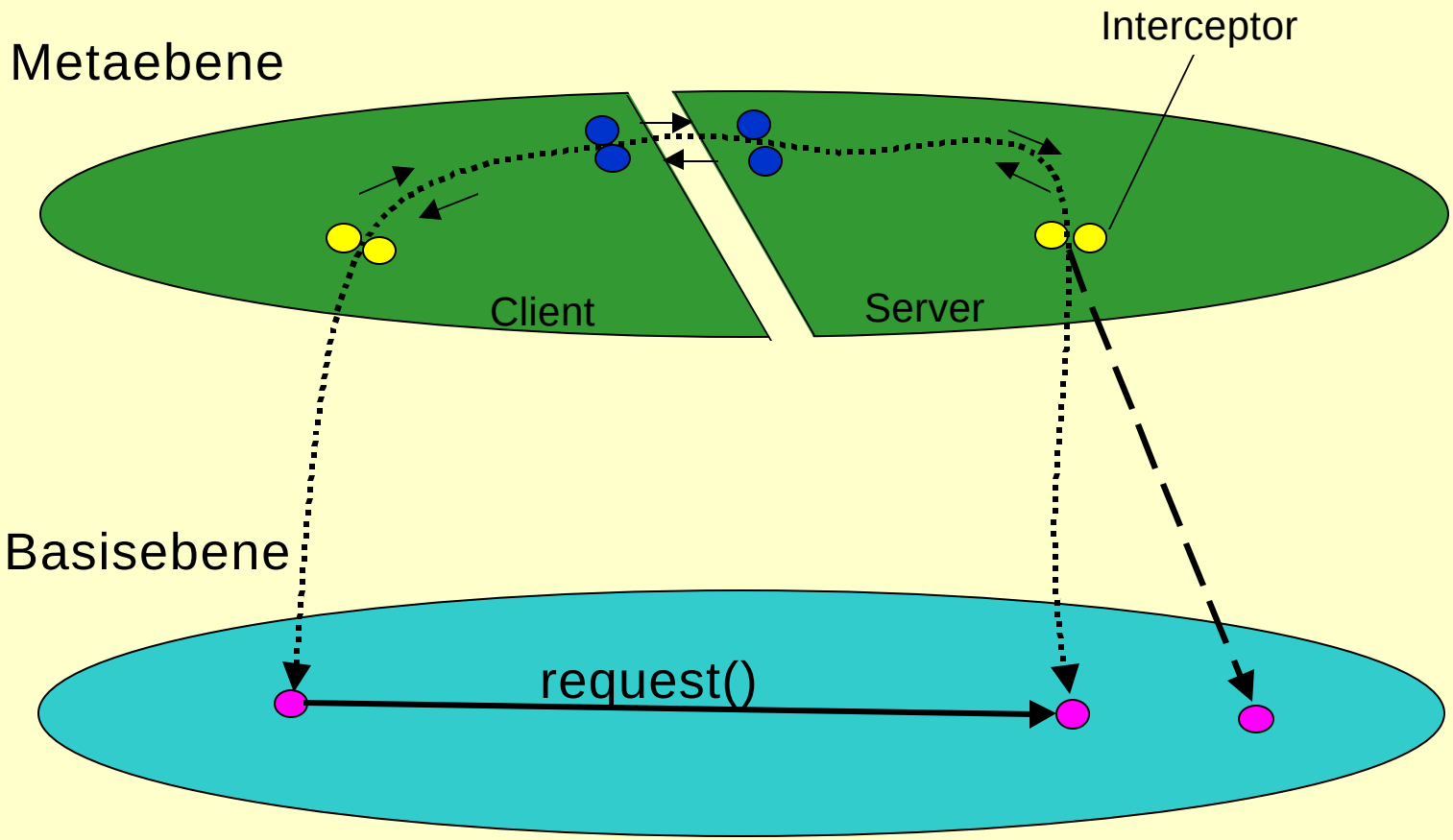
- Interceptors und DII

  $\Rightarrow$ *meta-communication model*

- Laufzeit-Typinformation

  $\Rightarrow$ *meta-class model*

CORBA-Reflection I: Interceptors



Interceptors als Reflection-Konzept

- Implizite Reflexion
- reifizierte Nachricht als Argument

  $\Rightarrow$ *meta-communication model*
- Änderung der Nachricht, Umleitung, Ver-/Entschlüsselung, Protokollierung etc.

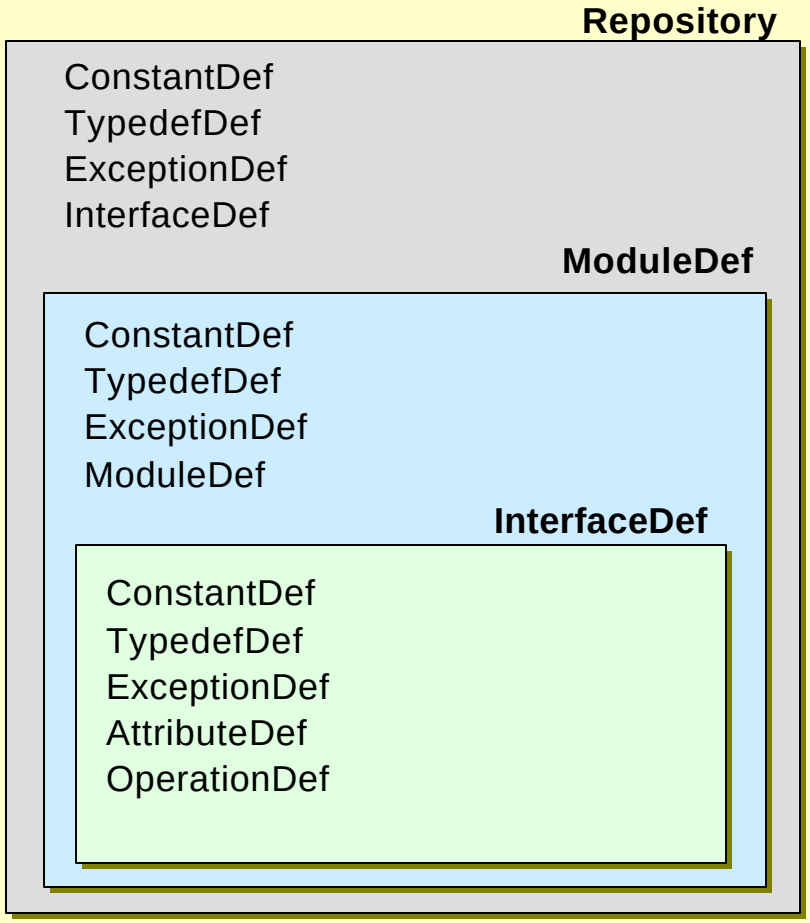
  $\Rightarrow$ *behavioral reflection*

CORBA-Reflection II: Laufzeit-Typinformation

- Typsicherheit trotz Offenheit und Heterogenität
- dynamische Aufrufe via DII (Gateways, etc.)

$\Rightarrow$ *Interface Repository*:
Datenbank mit Typinformation

CORBA Interface Repository



- Hierarchie von Behältern
- erlaubt Anfragen an das IDL-Typsystem
- *RepositoryIDs* identifizieren Typbeschreibungen:

`"IDL:jacorb/demo/grid:1.0"`

- Aufruf **`get_interface()`** liefert ein **`InterfaceDef`**-Metaobjekt aus dem IR

IR als Reflection-Konzept

- *Explizite* Reflexion
- Metaobjekte reifizieren Typen
  ⇒ *meta-class model*
- Inspektion, keine Modifikation
  ⇒ *structural reflection*

3. Reflection in Java

- Java-Reflection-API seit JDK 1.1
- kein *integraler* Sprachbestandteil
  - homogene Verteilung mit RMI, kein Bedarf für verteilte Typmanagement-Komponente
- Offenheit:
  - dynamisches Laden/Binden
- Plattformunabhängigkeit

Java Reflection-API

`java.lang.Class, java.lang.reflect.*`

- Laufzeittypinformation:
  `o.getClass(), Class, Field, Method`
- dynamische Objekterzeugung:
  `Constructor.newInstance(),`
  `Class.newInstance()`
- dynamische Aufrufe:
  `Method.invoke()`

Java-Reflection classified

- *Computational Reflection:*

  meta-class model,

  explicit,

  structural reflection

- *Implementational Reflection*:

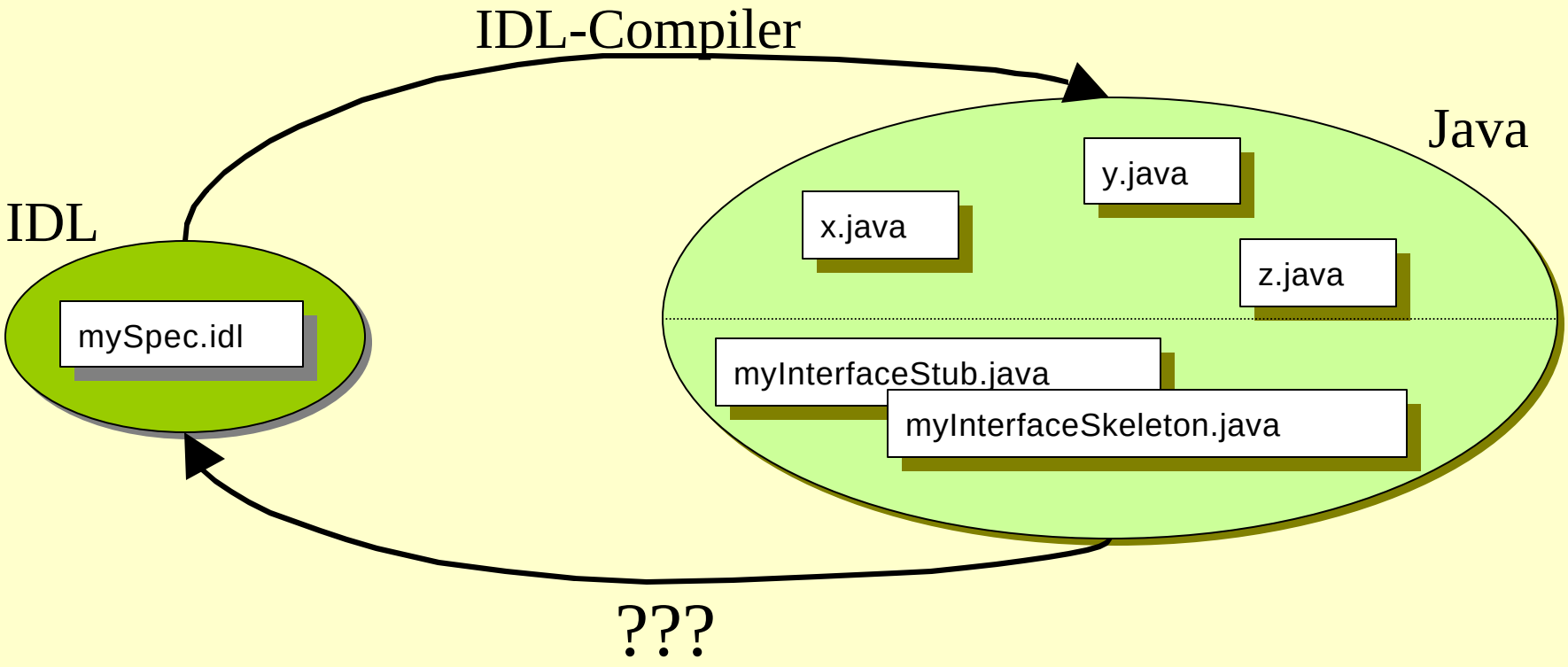
  `Thread, System, ClassLoader, Runtime, SecurityManager,` ...

4. JacORB

- freie CORBA 2.0-Implementierung
- vollständig in Java implementiert
- native IIOP, DII/DSI, Interceptors, Multithreading, Name- u. Event-Services
- “reflective” Interface Repository

http://www.inf.fu-berlin.de/~brose/jacorb

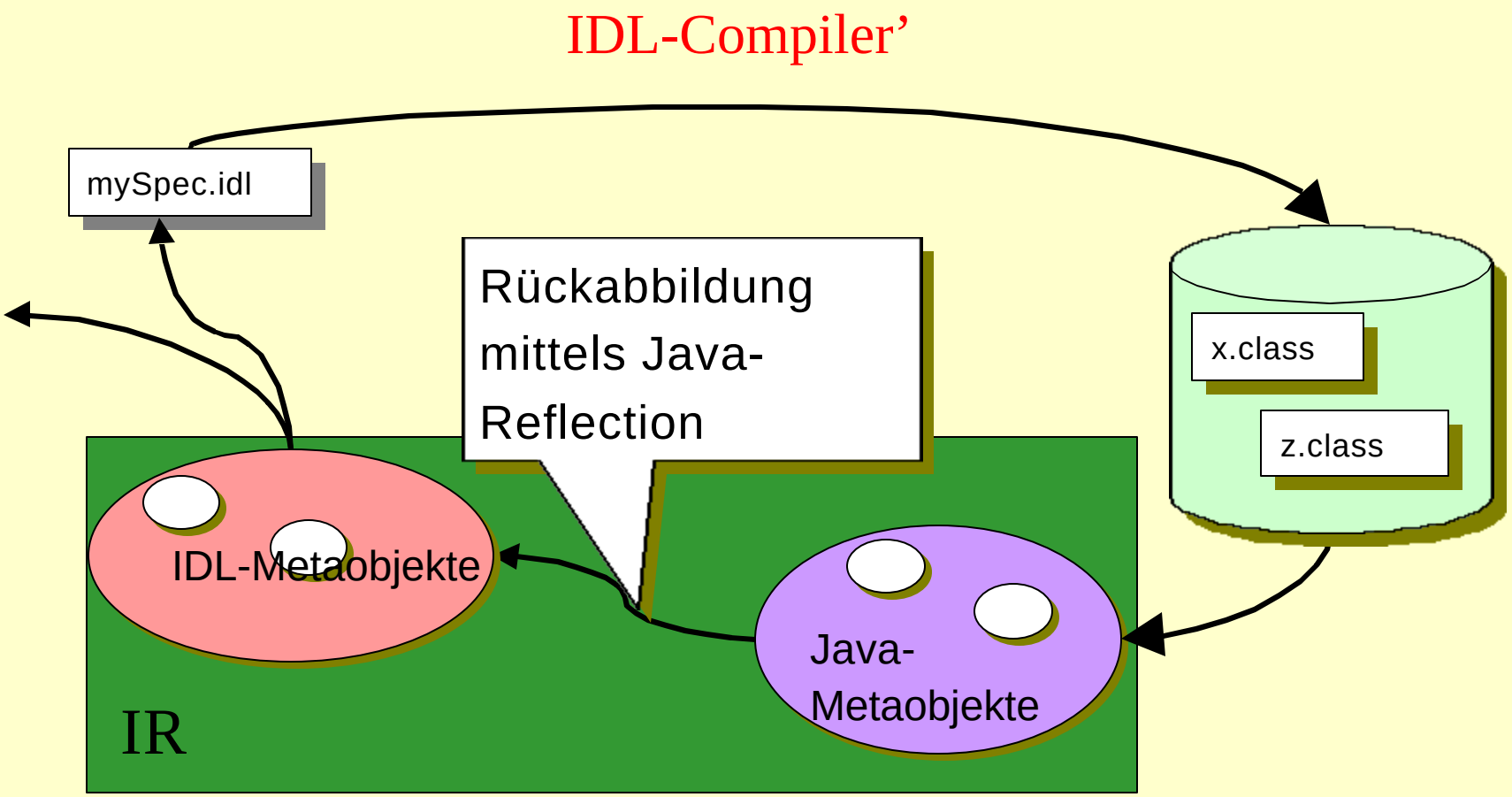
Von IDL nach Java - und zurück?



Interface Repository Design

- Idee: “CLASSPATH = IR-Inhalt”:
- Vorteile:
  - einfaches Repository
  - keine Redundanz
  - CORBA-Programmierung ohne IDL!
- Nachteile:
  - leicht modifizierte IDL-Java-Abbildung
  - keine Manipulation des Repository
  - keine Versionierung
  - kein **`#pragma prefix`** in IDL

JacORB IR


IDL-Compiler’
mySpec.idl
Rückabbildung mittels Java-Reflection
x.class
z.class
IDL-Metaobjekte
Java-Metaobjekte
IR

Zusammenfassung

- Reflection ermöglicht Offenheit
- verwandte Reflection-Konzepte in Java und CORBA
- Java-Reflection-API erlaubt elegante Implementierung des CORBA-IR
- konzeptionelle Nähe von CORBA und Java erlaubt CORBA-Programmierung ohne IDL!